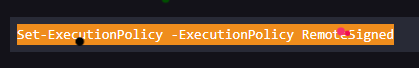
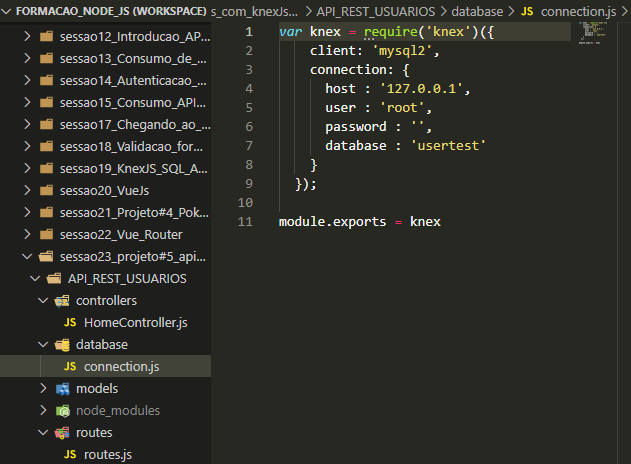
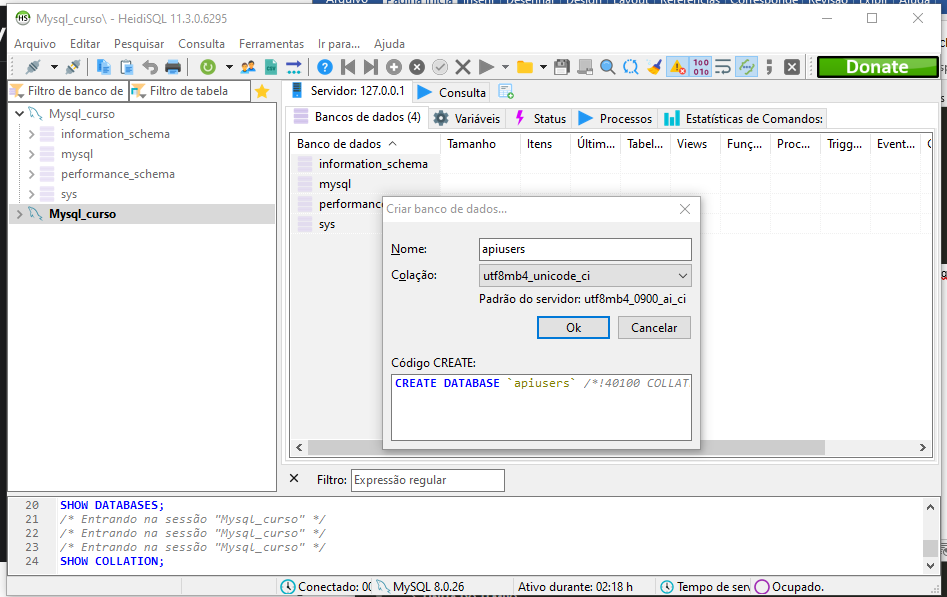
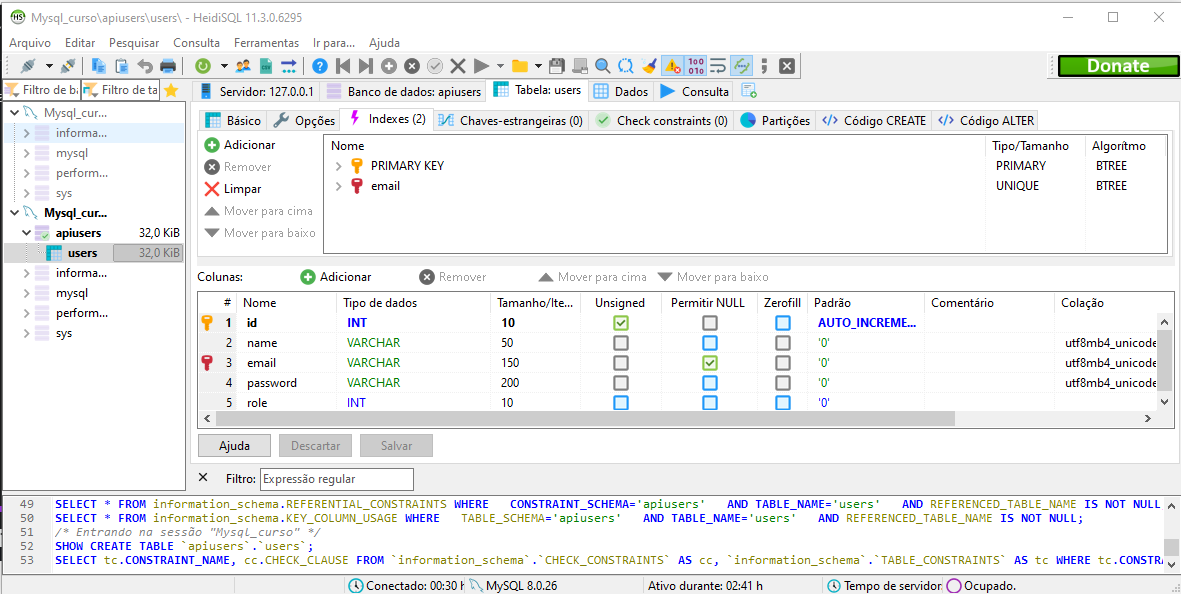
**SISTEMA DE GESTÃO DE USUÁRIOS**

1. Neste projeto iremos aprender a logica para implementação de um sistema de gestão de usuário completo, que pode ser utilizado em qualquer projeto futuro. Com cadastro, alterações, recuperação de senha etc...
2. Para agilizar a criação de um projeto foi criado a base de um projeto node no git hub para agilizar a criação
3. Link: github.com/techtuxbr/express-js-rest-api-base
4. Baixar projeto como ZIP, descompactar na pasta do projeto que será criado
5. Entrar na pasta do projeto e digitar : npm install
6. No arquivo index.js estamos importando quatro módulos
7. - boyParser que é utilizado para receber dados json via post.
8. – express
9. – router
10. Todas as rotas da aplicação serão definidas na pasta routes
11. Todos os códigos que se comunicam com o banco de dados será construído dentro da pasta models.
12. Para permitir código no powershell do Windows entre no powershell no modo administrador e altere a politica de segurança digitando:
13. 

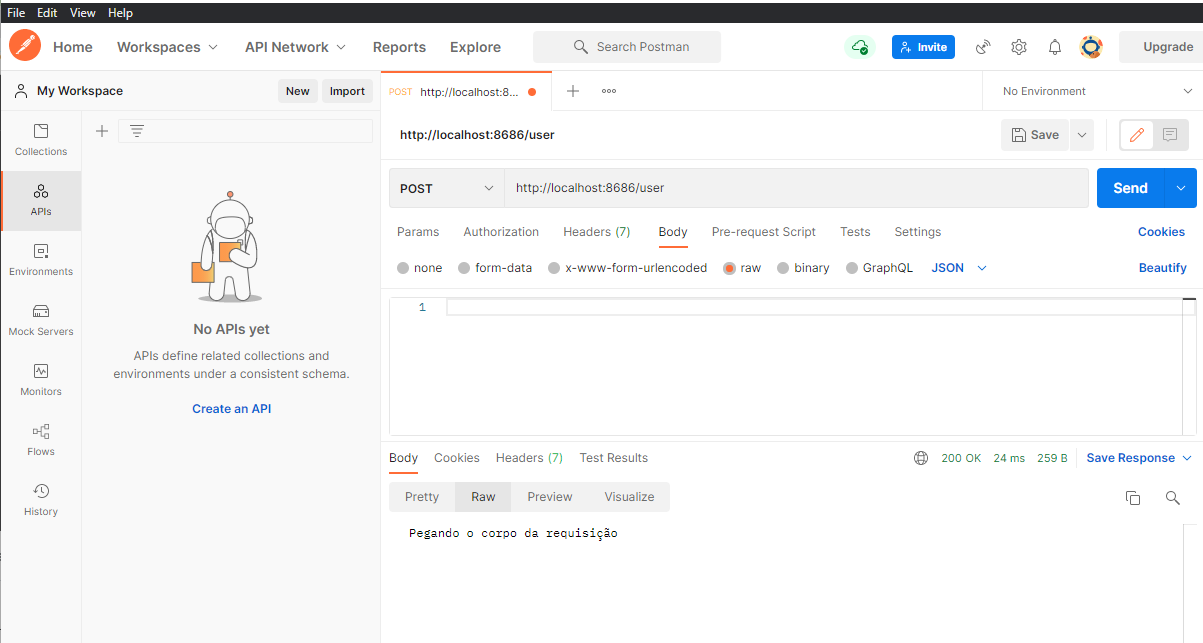
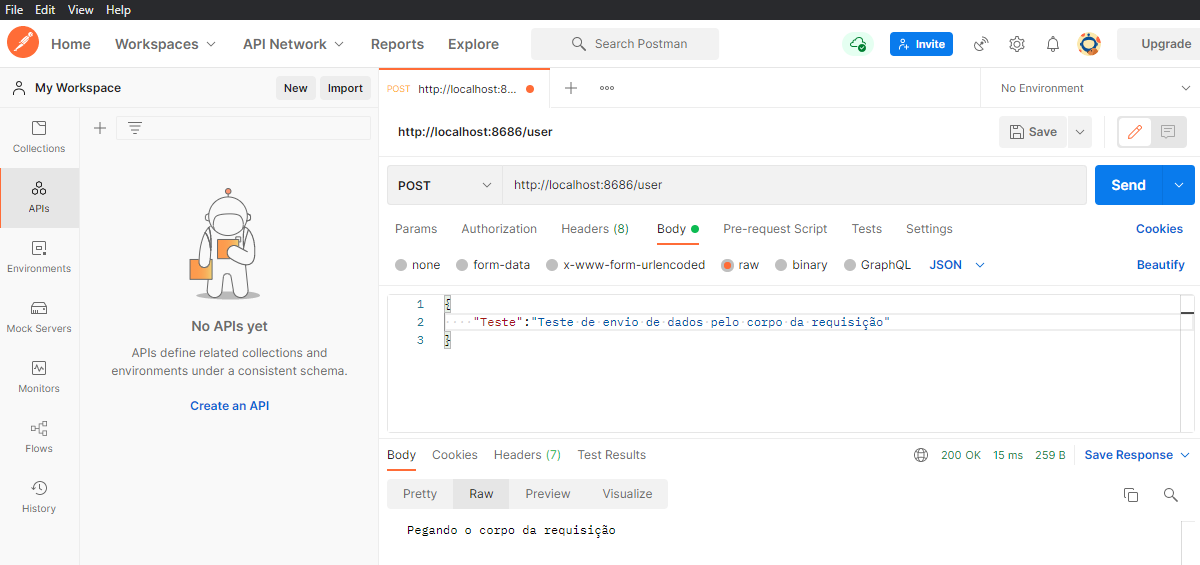
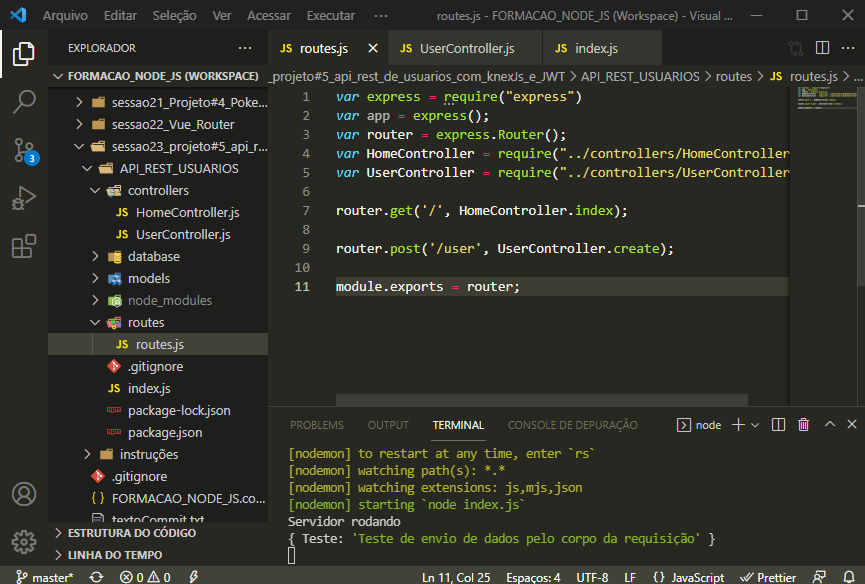
**Instalando o MySql**

1. Navegue até á a página mysql.com entre em Downloads, clique em “MySQL Community (GPL) Downloads”
2. Clique em “MySQL Community Server”
3. Selecione o sistema operacional e clique em “Go to Download Page”
4. Baixe a versão completa.
5. Após baixar clique no arquivo para instalar.
6. Quando abrir o programa, selecione a opção “Server Only” e next
7. Clique em “execute” para finalizar a instalação
8. Clique em next para iniciar a configuração.
9. Clique em next novamente
10. Primeiramente selecine “Develelopment Compute” em Config type.
11. Clieque em next
12. Cadastre um senha e next
13. Clique em next.
14. Clique em execute
15. Clique em finish
16. Clique em next
17. E Finish
18. Vamos linkar o executável do mysql no cmd
19. Vá até a pasta mysql em arquivos de programas
20. Entre na pasta bin
21. Copie o endereço desta pasta
22. Vá em painel de controle, sistema e segurança, sistema, propriedades do sistema, variáveis de ambiente,
23. Em variáveis de usuário para ...
24. Clicke em Path, editar, novo, colar o link que foi copiado, clique em ok , ok e ok.

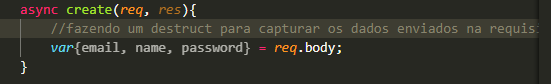
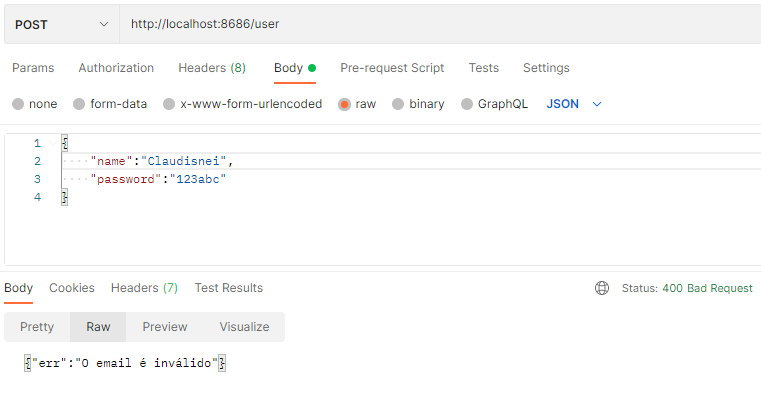
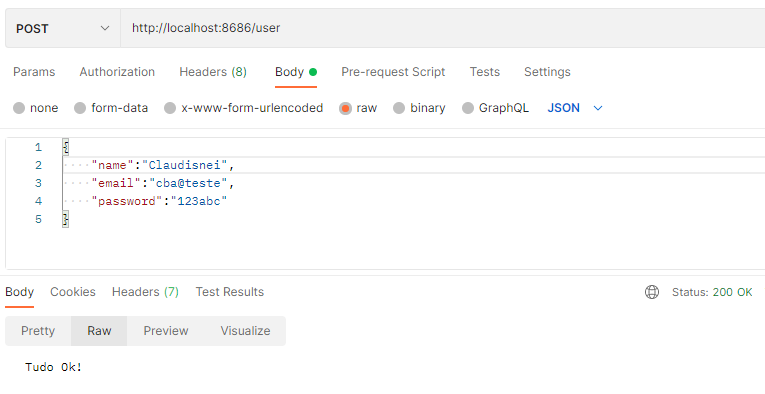
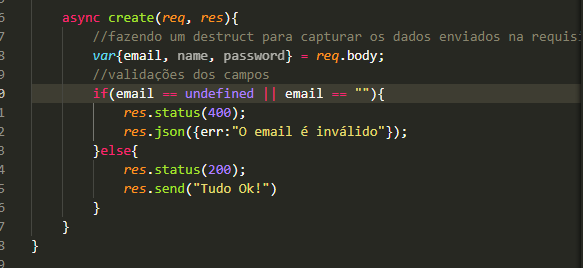
**Começando a crianção com o banco de dados**

1. Sempre que vc for criar uma aplicação com query builder como o Knex, temos que começar com a definição do banco de dados. Vamos utilizar o SGBD HeidiSql.
2. Antes vamos abrir o arquivo de configuração da conexão do banco de dados
3. 
4. Substituía os valores no arquivo connection, pelos dados de sua conexão com o mysql2
5. 
6. Após isso vá no SGBD, neste caso no HeidiSQL e crie um banco de dados
7. 
8. Agora vamos definir a única tabela da api chamada users
9. Crie os campos :
10. 

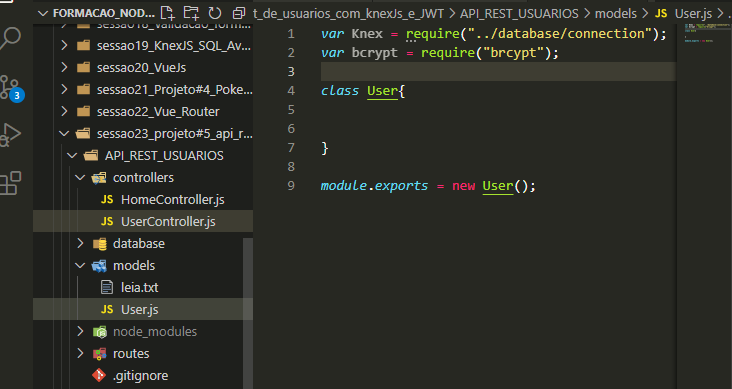
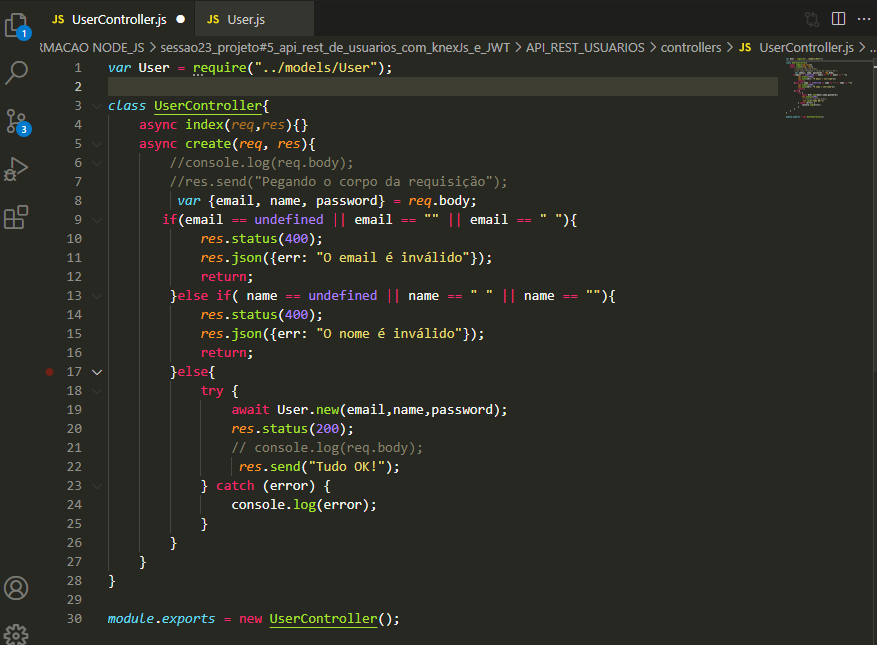
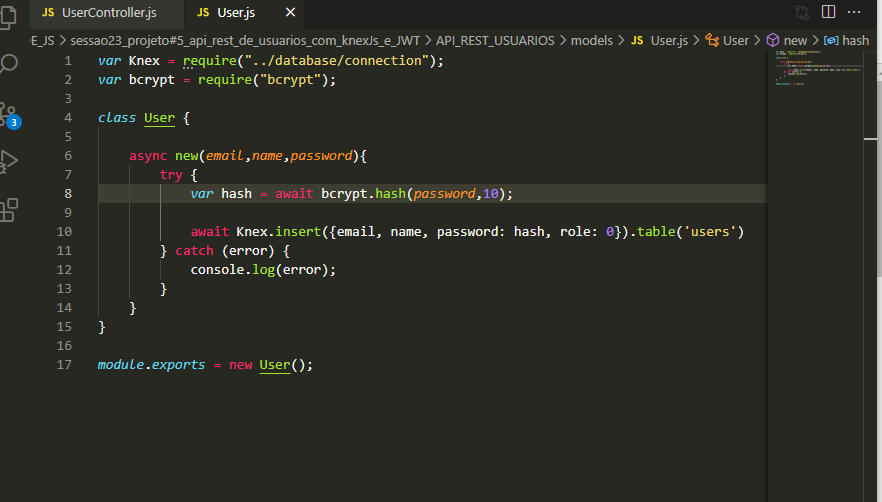
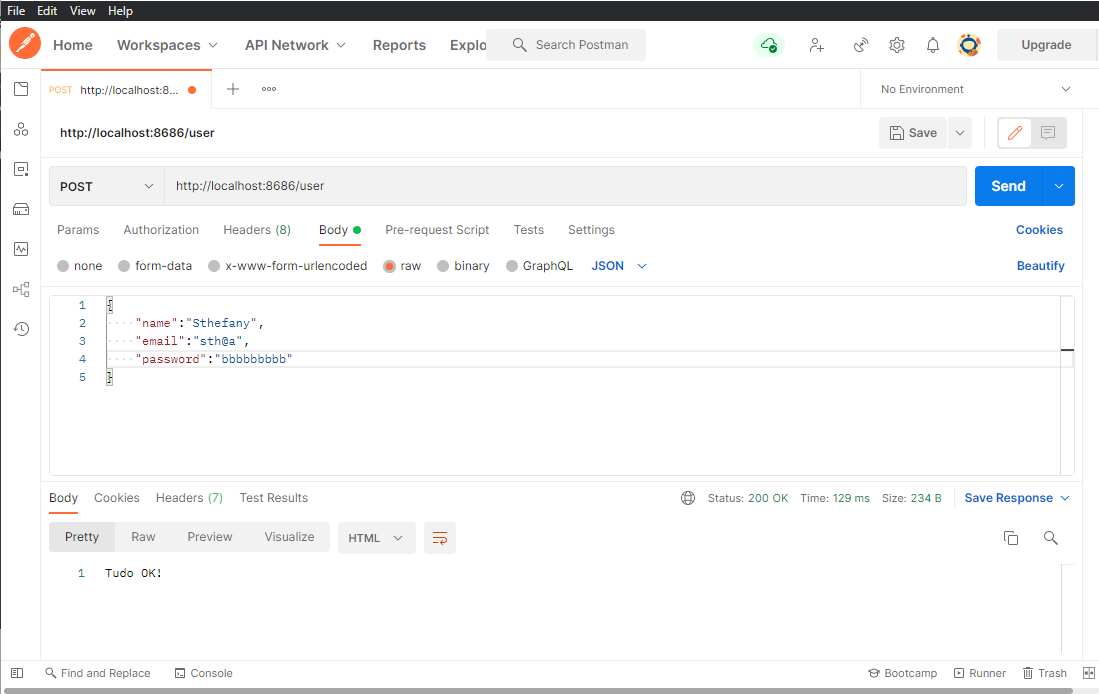
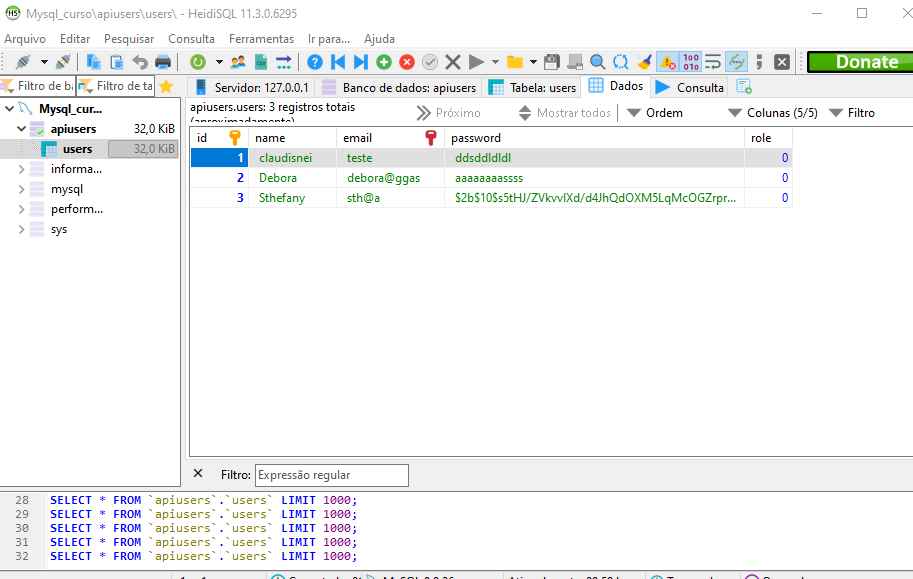
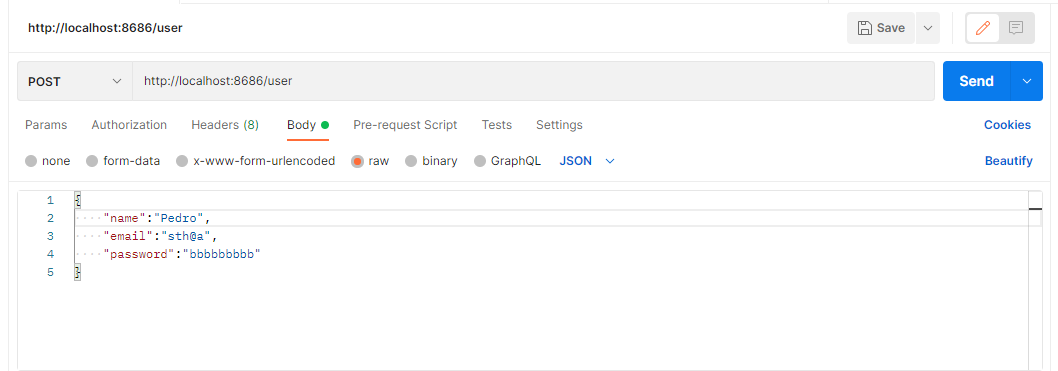
**Se adaptando a estrutura dos Controllers**

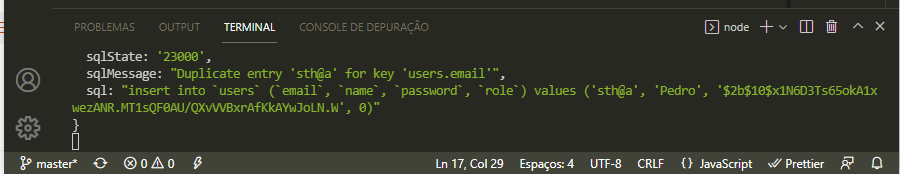
1. na pasta controllers, crie o arquivo UserController.js para criar a lógica de acesso aos dados da aplicação
2. Lembrando que todo end point da aplicação deve ter uma resposta ‘res’
3. No fim do arquivo crie um module.exports para exportar o UserController().
4. 
5. Para usar o controller temos que importa-lo no arquivo de rotas e criar a rota em seguida. Neste caso vamos criar uma rota post já que vamos criar um novo usuário.
6. Router.post(‘/user’), o hast entende que como a rota é post e é passado somente o user, a rota será para salvar um novo usuário.
7. Para testar rotas post, temos que trabalhar om o Postman, por isso vamos instala-lo.
8. Após isso rode novamente o servidor
9. Vá no postman e digite a url <http://localhost:8686/user> na rota post.
10. Aparecerá a mensagem que configuramos no método create no UserController.
11. 
12. Para testar se os dados estão sendo enviados pelo corpo da páigna, no Postman, clique em Body, raw, selecione o formato Json. Escreva uma mensagem em Json. E clique em send.
13. 
14. No console do Vscode aparecerá a mensagem que foi enviada.
15. 

Validação de dados

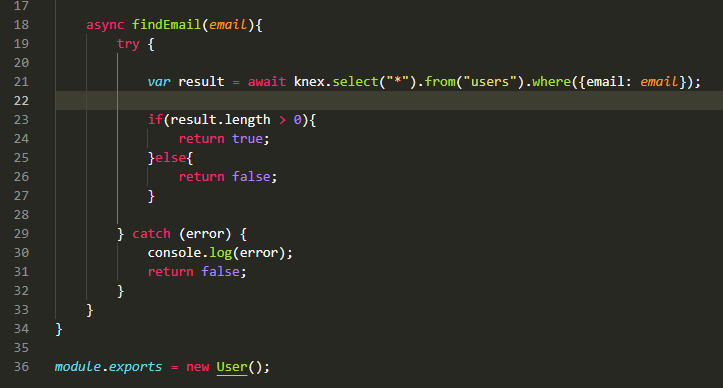
1. Para evitar que o usuário envie dados desnecessários e incorretos para uma rota, os mesmos serão validados.
2. Já foi falado de validação de formulários no curso, por isso vamos fazer uma validação simples.
3. No método create do UserController vamos fazer um destruct, criando três variáveis que receberão os dados enviados pelo corpo da requisição.
4. 
5. Na sequência verificamos se o email é undefined, neste caso será enviado o status code 400 - bad request.
6. 
7. E caso contrário será enviado a resposta de status code 200 - ok
8. 
9. O código ficará assim:
10. 

**Trabalhando com banco de dados**

1. Iremos receber dados via requisição para cadastrar um usuário no banco, porém precisamos criptografar a senha.
2. Não podemos salvar a senha do usuário de forma plana, a maneira mais segura é utilizando um hash temos então que transformar a senha em um hash.
3. Para trabalhar com hash no nodejs, vamos instalar a biblioteca bcrypt.
4. Npm install bcrypt - -save.
5. Vamos trabalhar com models utilizando o knex para acessar banco de dados.
6. Vamos criar um model chamado User.js que será uma classe que terá os métodos dela
7. Na classe User vamos carregar o Knex e o bcrypt. O knex irá se conectar ao banco de dados utilizando o arquivo connection da pasta database e bcrypt irá gerar um hash da senha do usuário.
8. 
9. Esse model terá uma função assíncrona chamada new() que receberá um email, password e name para cadastrar, chamaremos o knex.insert passando email, password, name e role como parâmetros.
10. Role por padrão será zero ou seja todo usuário novo será do tipo comum.
11. Será passado ainda como parâmetro dentro de table() a tabela onde os dados serão gravados, neste caso users.
12. Vamos criar um try e cath e colocar await antes de knex para tornar a função assíncrona.
13. 
14. Vamos importar o model User dentro do controler UserController.
15. Após a validação utilizaremos o model User com o método new e os parâmetros email, name e password. Temos que utilizar await pois é uma função assíncrona. Por ser um método dentro de um model, temos que utiliza try para capturar possíveis erros.
16. 
17. O usuário está sendo salvo no banco de dados corretamente, vamos agora utilizar o bcrypt para esconder a senha do usuário em um has.
18. A documentação do bcrypt na internet ensina como utiliza-lo com promisses.
19. Vamos importar o bcrypt no model User.
20. O bcrypt irá chamar a função hash que receberá como parâmetros a senha do usuário e um número que será a quantidade de vezes que a senha será “hasheada”.
21. Vamos criar uma variável hash que irá receber o hash criado pelo bcrypt. O password passado no Knex receberá o hash.
22. 
23. Para testar vamos mandar uma requisição no Postman com os dados de um usuário.
24. 
25. O resultado na tabela users será:
26. 
27. Se tentarmos salvar um novo usuário com um email já cadastrado
28. 

1. irá aparecer o seguinte erro no console, pois foi configurado no banco de dados que o email seria uma chave do tipo Unique, ou seja não pode haver no banco de dados dois email iguais.
2. 

**Método findEmail**

1. Vamos criar um novo método no controller User, chamado findEmail para, antes de salvar no banco de dados, fazer uma busca no BD e verificar se o email já está cadastrado.
2. Vamos criar uma variável result que irá receber o resultado da busca realizada pelo Knex, sendo passado para ele uma string sql. Caso haja um email idêntico já cadastrado o result.length será maior que zero então teremos o retorno como true, caso contrário termos o retorno como false.
3. 
4. No controller UserController vamos criar uma variável emailExists que receberá o resultado do método findEmail.
5. Se o email existir iremos retornar um res.status(406) – Não aceitável e encerraremos a requisição.
6. Caso contrário iremos cadastrar o usuário.
7. 